代码示例

四邻域填充法的核心代码如下。

```
const int dx[4] = {-1,0,0,1};
const int dy[4] = {0,-1,1,0}; // 以四邻域填充法为例
// newcolor 表示替换颜色 oldcolor 表示目标颜色
void dfs(int x,int y,int newcolor,int oldcolor)
{
    c[x][y] = newcolor;
    for (int i = 0; i < 4; ++i)
    {
        int nx = x + dx[i],ny = y + dy[i];
        if (nx < 1 || nx > n || ny < 1 || ny > n)
            continue;
        if (c[nx][ny] != oldcolor)
            continue;
        dfs(nx,ny,newcolor,oldcolor);
    }
}
```

参考词条

1. 深度优先遍历
2. 广度优先遍历

（李绍鸿　谢秋锋）

1.4.8　动态规划

1.4.8.1　动态规划的基本思路

动态规划是 1957 年理查德·贝尔曼在 *Dynamic Programming* 一书中提出的一种表格处理方法，它将原问题分解为若干子问题，自底向上先求解最小的子问题，并把结果存储在表格中，在求解大的子问题时直接从表格中查询小的子问题的解，以避免重复计算，从而提高效率。

动态规划算法常用来求解最优化问题，尤其是带有多步决策的最优化问题。能用动态规划解决的问题具备以下 3 个要素。

（1）最优子结构：如果问题的最优解所包含的子问题的解也是最优的，就称该问题具有最优子结构。也就是说一个问题的最优解只取决于其子问题的最优解。

（2）无后效性：将原问题分解为若干子问题，每个子问题的求解过程作为一个阶段，当前阶段的求解只与之前阶段有关，与之后阶段无关，即某阶段的状态一旦确定，

就不受这个状态后续决策的影响。

（3）重叠子问题：求解过程中每次产生的子问题并不总是新问题，会有大量子问题重复。在遇到重复子问题时，只需在表格中查询，无须再次求解。该性质不是使用动态规划解决问题的必要条件，但是凸显了动态规划的优势。

动态规划解题的一般设计模式如下。

（1）划分阶段：按照问题的时间或空间特征，将问题分为若干个阶段。这也是动态规划状态转移的顺序，所以在划分阶段时，要注意划分后的阶段一定要有序或者是可排序的，以保证各状态的无后效性。

（2）状态表示：将问题发展到各个阶段时所处于的各种情况用不同的状态表示出来，通常状态的表示可以设为问题最终结果的一般化表示，并将最优解进行递归定义。

（3）决策与状态转移方程：在对问题的处理中做出的每种选择性的行动称为决策，即从该阶段的每一个状态出发，通过一次选择性的行动转移至下一阶段的相应状态。根据上一阶段的状态和决策来导出本阶段的状态就是状态转移。通常做法是根据相邻两个阶段的状态之间的关系来确定决策方法和状态转移方程。

（4）边界条件：给出的状态转移方程是一个递推式，需要一个递推的终止条件或边界条件。

（5）答案：问题的求解目标。

由数字组成的金字塔型图案如图 1.12 所示：

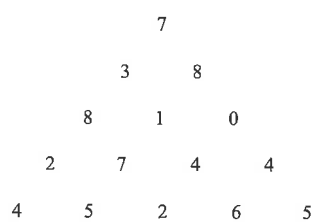

图 1.12　由数字组成的金字塔型图案

问题（IOI1994 Number Triangle）：从最高点到底层任意处结束有多条路径，每一步可以走到左下方的点也可以到达右下方的点，求路径上经过数字的和的最大值。

这个问题可以看作多步决策求最优解，即按层逐层做最优决策，整个求解过程符合动态规划求解的 3 个要素，可用动态规划思想求解，算法设计如下。

（1）状态：$dp_{i,j}$ 表示第 i 行第 j 列走到最底层时经过的数字的最大和。

（2）状态转移方程：$dp_{i,j}=\max\{dp_{i+1,j},dp_{i+1,j+1}\}+a_{i,j}$，其中 $a_{i,j}$ 为数字三角形中第 i 行第 j 列的数值。

（3）边界：$dp_{n,1}=a_{n,1},dp_{n,2}=a_{n,2},\cdots,dp_{n,n}=a_{n,n}$。

（4）答案：$dp_{1,1}$。

本算法的时间复杂度为 $O(n^2)$。

代码示例

动态规划解决 IOI1994 Number Triangle 问题的核心代码如下。

```
for (int i = 1; i <= n; i++) dp[n][i] = a[n][i]; // 初始化最底层状态的状态值
for (int i = n - 1; i >= 1; i--)
{ // 自底向顶计算
    for (int j = 1; j <= i; j++)
        dp[i][j] = max(dp[i + 1][j],dp[i + 1][j + 1]) + a[i][j];
    // 枚举两种可能的决策方式进行转移
}
printf("%d",dp[1][1]); // 输出答案
```

参考词条

1. 递归法
2. 记忆化搜索

延伸阅读

THOMAS H C，CHARLES E L，RONALD L R，et al. 算法导论(原书第 3 版)[M]. 殷建平，徐云，王刚，等译. 北京：机械工业出版社，2013：204-210.

(金靖　谢秋锋)

1.4.8.2　简单一维动态规划

一维动态规划是在一维数组上标识状态及进行转移的动态规划。

最长上升子序列(LIS)是一个常见的使用一维动态规划来解决的问题。最长上升子序列问题是对于一个给定的长度为 n 的序列，求其单调递增的最长子序列的长度。算法设计如下(阶段按每个数来划分)。

(1) 状态：$dp[i]$表示以 $a[i]$结尾的最长上升子序列的长度。

(2) 状态转移方程：$dp[i]=\max\limits_{0\leqslant j<i,a[j]<a[i]}\{dp[j]+1\}$。对于位置 i 之前的每一个位置 j，如果满足 $a[j]<a[i]$，都可以将 $a[i]$接在以 $a[j]$结尾的子序列后面得到一个长度加 1 的子序列，其中最长的一个即为以 $a[i]$结尾的最长上升子序列。

(3) 初始状态：$dp[0]=0$。

(4) 答案：$\max\limits_{1\leqslant i\leqslant n}(dp[i])$。

本算法的时间复杂度为 $O(n^2)$。

代码示例

求最长上升子序列(LIS)的核心代码如下。

```
dp[0] = 0; // 边界
res = 0; // 答案的初始值
for (int i = 1; i <= n; i++)
{
    dp[i] = 1;
    for (int j = 1; j <= i - 1; j++)
    {
        if (a[j] < a[i]) // 寻找能接且最长的子序列
            dp[i] = max(dp[i],dp[j] + 1);
    }
    if (dp[i] > res)
        res = dp[i]; // 更新最大值
}
```

参考词条

动态规划的基本思路

典型题目

1. IOI1994 Number Triangles
2. IOI1999 花店橱窗布置
3. NOIP1996 提高级 挖地雷
4. NOIP1999 普及组 导弹拦截
5. NOIP2000 提高级 方格取数
6. NOIP2004 提高组 合唱队形
7. NOIP2008 提高组 传纸条
8. NOIP2010 提高组 乌龟棋
9. NOIP2012 普及组 摆花
10. NOIP2013 提高组 花匠
11. NOIP2015 提高组 子串
12. CSP2020-J 方格取数
13. CSP2022-J 上升点列

（谢秋锋）

1.4.8.3 简单背包类型动态规划

背包问题是动态规划的经典问题之一。背包问题的一般形式是，有 n 种物品和一个容量为 V 的背包，第 i 种物品的费用（如体积）是 v_i、数量是 n_i、每个价值是 c_i，可以选择任意给出的物品中的若干个放入背包中。求满足所选择物品的体积之和不超过 V 的情况下物品价值的和最大是多少？当每种物品数量为 1 时，该问题称为 0-1 背包问题。

0-1 背包问题可以看作一个多步决策问题。若不选择第 i 个物品，则需要在前 i-1

个物品中选择若干个使得体积和不超过 V 的物品；若选择第 i 个物品，则需要在前 $i-1$ 个物品中选择若干个使得体积和不超过 $V-v_i$ 的物品。最优子结构性质是显然的，而由于候选的物品数量在不断减少，故无后效性也成立。因此可以使用动态规划算法解决该问题，算法设计如下。

（1）状态：dp[i][j]表示前 i 种物品放入容量为 j 的背包中所获得的最大价值。

（2）状态转移方程：$dp[i][j]=\begin{cases} dp[i-1][j], & j<v[i] \\ \max\{dp[i-1][j], dp[i-1][j-v[i]]+c[i]\}, & j\geqslant v[i] \end{cases}$

（3）初始值：dp[ ][ ]数组的 0 行 0 列为 0，即 dp[0][j]=0，dp[i][0]=0，其中 i=0,1,…,n，j=0,1,…,V，表示处理第 0 种物品或背包容量为 0 时，获得的价值均为 0。

（4）答案：dp[n][V]。

本算法的时间复杂度和空间复杂度均为 $O(nV)$。

代码示例

0-1 背包问题的核心代码如下。

```
for (int i = 1; i <= N; i++) // 依次枚举要考虑的物品
    for (int j = 0; j <= V; j++)
        if (j >= v[i])
            dp[i][j] = max(dp[i - 1][j],dp[i - 1][j - v[i]] + c[i]);
            // 能选第 i 个物品时,在是否选择间做最优决策
        else
            dp[i][j] = dp[i - 1][j]; //不能选第 i 个物品
printf("%d",dp[N][V]); //输出答案
```

参考词条

动态规划的基本思路

典型题目

1. NOIP2001 普及组 装箱问题
2. NOIP2005 普及组 采药
3. NOIP2006 普及组 开心的金明
4. NOIP2006 提高组 金明的预算方案
5. NOIP2014 提高组 飞扬的小鸟
6. NOIP2018 提高组 货币系统
7. CSP2019-J 纪念品

（谢秋锋）

1.4.8.4 简单区间类型动态规划

区间动态规划是在区间上进行动态规划，即求解一段区间上的最优解。与一维动态